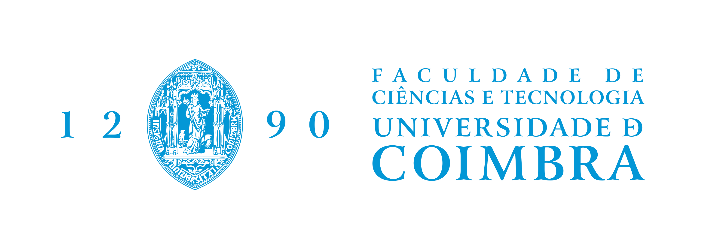
******

**Relatório do Projeto**

*Data Serialization and Deserialization with XML and Google Protocol Buffers*

**Integração de Sistemas**

**2019/2020**

Duarte Guerreiro 2016231778

João Tomás 2016225021

**Índice**

[1. Introdução 3](#_Toc20923933)

[2. Como executar o projeto 3](#_Toc20923934)

[3. Arquitetura do projeto 3](#_Toc20923935)

[4. Ficheiros constituintes do projeto 4](#_Toc20923936)

[5. Condições da experiência 5](#_Toc20923937)

[6. Testes efetuados 6](#_Toc20923938)

[7. Conclusões sobre os testes efetuados 6](#_Toc20923939)

# **Introdução**

Este trabalho foi realizado com o intuito de comparar duas das tecnologias existentes para serialização e desserialização de dados, sendo estas **XML (Extensible Markup Language),** como representação de texto, e **Google Protocol Buffers**, como representação binária.

Para que tal fosse possível, foram criados tanto um cliente como um servidor, usando tecnologia **gRpC** (sistema open source para chamadas remotas de métodos, inicialmente desenvolvido pela Google).

Não obstante, achamos também importante referir a utilização do Maven para gerir o nosso projeto. Este utiliza o ficheiro pom.xml de forma a descrever o projeto que está a ser construído, as suas dependências, a ordem de compilação e os plugins necessários.

**Descrição das tecnologias utilizadas (XML, Google Protocol Buffers e gRPC)**

# **Como executar o projeto**

Para que seja possível correr o projeto é apenas necessário executar o ficheiro TheServer.java em primeiro lugar e, de seguida, o ficheiro Client.java. É necessário que esta ordem seja respeitada visto que, se o cliente correr primeiro do que o servidor, este vai tentar ligar-se a um servidor que não se encontra ativo e vai dar origem a uma exceção.

# **Arquitetura do projeto**

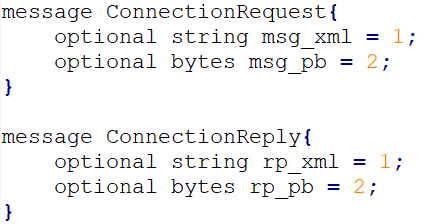
****O nosso projeto é constituído por um cliente e um servidor, que comunicam entre si utilizando a tecnologia gRPC e trocam dados serializados, através da utilização das tecnologias XML e ProtocolBuffers. Para que o cliente e o servidor possam comunicar, é definido, no ficheiro **grpcp.proto**, um serviço gRPC e os métodos de pedido e resposta, no qual o cliente envia um pedido usando o stub e espera que a resposta do servidor regresse.

Figura 1 - grpcp.proto

Neste ficheiro encontra-se também definido os tipos de dados que serão trocados entre o cliente e o servidor, sendo que, para o uso de XML foi definido string visto que os dados vão ser trocados nesse formato, e para Protocol Buffers foi definido bytes porque é nesse formato que vão ser serializados.

Quanto ao funcionamento usando **Google Protocol Buffers**, começamos por criar o ficheiro .proto (**carownerpb.proto**) que irá conter as diferentes estruturas de dados necessárias, sendo estas Car, Owner, CarList e OwnerList, cada uma destas definida como uma *Message*. As duas primeiras estruturas têm a si associadas diversos campos, sendo que em Car temos id, brand, model, engine\_size, power, consumption, plate, owner\_id e em Owner temos id, name, telephone e address. Todos os campos possuem o modificador *required*, que significa que é obrigatório fornecer um valor para cada um desses campos. As duas últimas estruturas têm os seus atributos associados ao modificador *repeated,* uma vez que pretendemos criar tanto um array de owners como de carros, que serão posteriormente serializados. Após a definição deste ficheiro, corremos o compilador de Protocol Buffers *protoc*, o que irá permitir gerar todas as classes necessárias para escrever as mensagens a ser trocadas. Feito isto, passamos à criação dos métodos necessários(usando o Builder) para a criação da lista de donos que irá posteriormente ser serializada para um array de bytes (usando o método *toByteString)* e enviada para o servidor, onde será desserealizada (usando o método *parseFrom)* e analisada de forma a retornar a lista de carros por dono, que irá ser serealizada e enviada ao cliente como resposta ao seu pedido.

Relativamente ao funcionamento do **XML**, começámos por criar as classes relativas às estruturas de dados que vão ser usadas (Carro.java e Dono.java). Para que seja possível enviar os conjuntos (arraylists) de donos e de carros foi necessário criar uma wrapper class (ListaCarro.java e ListaDono.java), que irá permitir fazer a devida serialização do objeto para XML. Para efetuar a serialização do objeto Java para XML criámos a classe ObjectToXml, que possui dois métodos, um para serealizar a lista de carros e outro para a lista de donos. O ficheiro XML é criado usando **JAXB** (Java Architecture for XML Binding), que é uma API que fornece um mecanismo que para fazer marshall (escrever) objetos Java para XML e unmarshall (ler) XML para objeto Java. Para que o XML criado ficasse formatado e não numa única linha usámos o método *setProperty*. Após o objeto ter sido convertido para XML, é convertido para string para que seja posteriormente enviado pelo cliente para o servidor, que irá converter a string para XML e fazer a sua desserialização de XML para objeto Java (esta operação fica a cargo da classe XmlToObject, onde se encontram os métodos para este efeito). Tendo o objeto Java, analisamos a lista de donos recebida e retornamos a lista de carros por dono, que irá ser serealizada e convertida para string para que seja enviada ao cliente como resposta ao seu pedido.

# **Ficheiros constituintes do projeto**

1. No package **java**:

* Client
* TheServer
* Carro
* Dono
* GetCars
* GetOwners
* ListaCarro
* ListaDono
* ObjectToXml
* XmlToObject

1. No package **proto**:

* carownerpb.proto
* grpcp.proto

1. Ficheiro **pom.xml**
2. Ficheiros **Car.txt** e **Owner.txt**

# **Condições da experiência**

Características do computador:

* Processador: Intel Core i7-7500U 2.70GHz 2.90GHz
* Memória Física (RAM): 16GB
* Placa Gráfica: Intel HD Graphics 620
* Sistema Operativo: Windows 10
* Tipo do sistema: x64-based PC

Tecnologias usadas**:**

* gRPC (versão 1.23.0)
* XML (versão 1.0)
* Google Protocol Buffers (versão 3.9.0)

Linguagens utilizadas:

* Java (versão 1.8.0\_221)

IDE utilizado:

* IntelliJ IDEA 2019.2.2

# **Testes efetuados**

**NOTA: Tamanho das estruturas de dados (Depois de serem serealizadas, tanto no XML como no Google Protocol Buffers)**

**NOTA: Incluir gráficos (plots), resta saber é com que dados em concreto**

**Gráficos:**

**- Tamanho das estruturas de dados em função do tempo e com as duas tecnologias usadas (linhas sobrepostas no mesmo gráfico ou gráficos distintos)**

**- Tamanho das estruturas de dados em função do número de donos e de carros**

**NOTA:** As medições do tempo foram efetuadas tanto no cliente como no servidor

# **Conclusões sobre os testes efetuados**

**NOTA: Colocar mais prints de partes relevantes do código**

**NOTA: Legendar os gráficos e as imagens**

**NOTA: Dizer os sítios do código onde medimos o tempo**